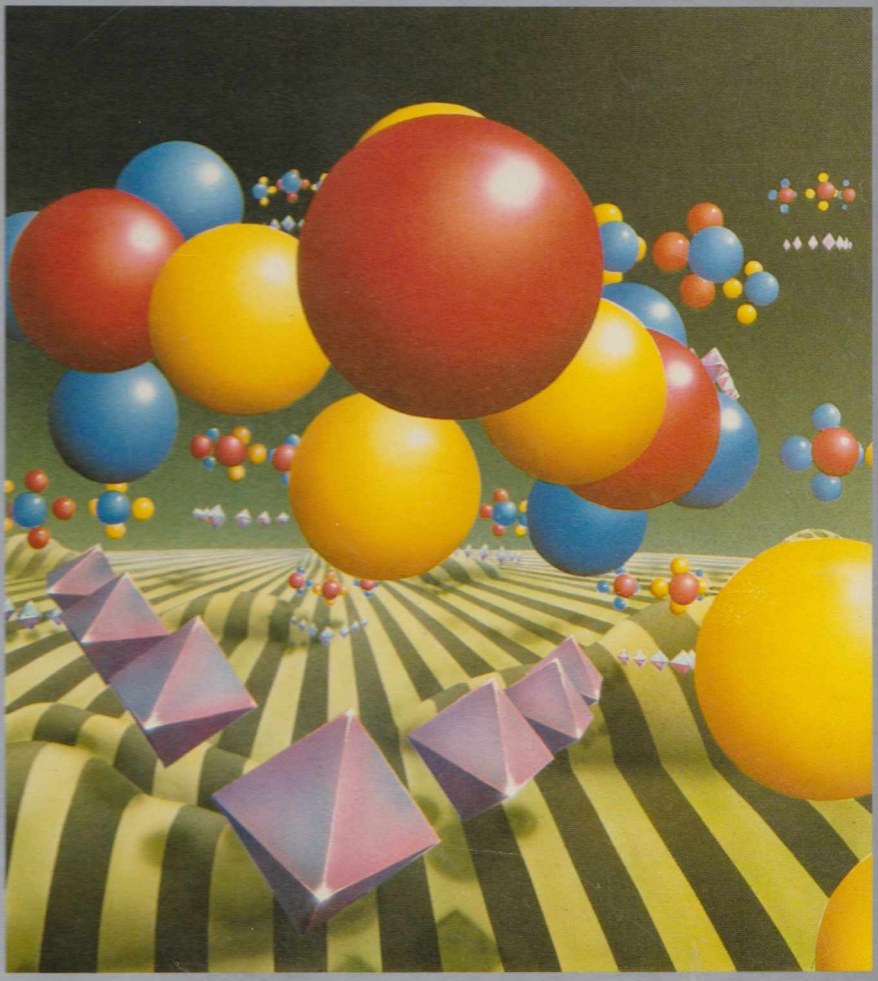
Visions

BREDEN'S BASIC

THE ESSENTIAL EXTENDED **BASIC** FOR
THE **COMMODORE 64** COMPUTER



COLOUR HIGH RESOLUTION GRAPHICS
MULTI COLOUR SPRITE GRAPHICS
MUSIC AND SOUND SYNTHESIS
STRUCTURED PROGRAMMING
USER DEFINED GRAPHIC CHARACTERS (UDGs)
TOOLKIT FACILITIES
PLUS MANY OTHER USEFUL FEATURES

VC — 12 — 64

DISK COMMANDS 6

ERROR TRAPPING 8

PROGRAMMING AIDS 9

ENHANCED PROGRAMMING STRUCTURES 10

APPENDICES 11

BREDEN'S BASIC

BREDEN'S BASIC

FOR THE COMMODORE 64

SIMON BREDEN

Visions

Visions (Software Factory) **Limited,**
1 Felgate Mews,
Studland Street,
W6 9JT

I am indebted to the following individuals for making this product possible. So THANKS to:

Jon Chisam
Geoff Barber
Alan Proctor
Robert Eatwell
Jayanti Shah
Sean de Bray
Charles Dunford
David Eastland
Chris Harding
Jeff Eldridge
Jeff Minter
Geoff Barber
Douglas Turner
Dave Hodgkinson
Victoria and William
Mum
Dad
Lord and Lady McIntosh of Haringey
Fran
Phil
Dave
Anne
Mandy
Dan
Judy
Anji
Derek Aslett
John and Simon MacNaught-Davis
Melissa and Angus
Douglas Turner

DISK

1/ Switch the disk drive and computer off, then on.
2/ Place the BREDEN's BASIC disk into the disk drive.
3/ Type:

LOAD "★",8,1

4/ Press the RETURN key.

CASSETTE

1/ Switch the computer off, then on.
2/ Place the BREDEN's BASIC cassette into the cassette unit.
3/ Hold down the SHIFT key and tap the RUN/STOP key.
4/ Press PLAY on the cassette unit.

CARTRIDGE

1/ Switch the computer off.
2/ Insert the BREDEN's BASIC cartridge into the computer.
3/ Switch the computer on.

N.B.
When removing or inserting cartridges, the computer MUST ALWAYS be switched OFF.

Conventions used in this manual

Within Breden's BASIC there are two types of Keyword. The first type is called a "command". Commands are used to instruct the computer to perform a specific task, such as changing the colour of the screen, for example.

Example of a command

SCOL 5
This changes the colour of the screen to green.
The other type of keyword supported is called a "function". Functions are used in expressions such as:
SCOL MOD(R,6)+4
The function, MOD, is used here to generate a value which can be used by the command, SCOL.
The value generated is called a "parameter" when it is used by commands such as SCOL in the example above.
Parameters, with Breden's BASIC, can be represented by algebraic expressions (AEXP) or string expressions (SEXP). Algebraic expressions can include binary expressions (BEXP) and/or hexadecimal expressions (HEXP). String expressions can consist of words in quotes and/or string variables.
In the parameters section of each keyword (see MOBSLOT, for example), the limitations are stated. For example, the value (algebraic) may have to be between 1 and 16.

This manual aims to teach you how all the keywords within Breden's BASIC function. It does not, however, attempt to teach you how the innards of the Commodore 64 work or how to program in BASIC. This can only be done with much reading and trial and error. We do strongly recommend you purchasing a book called the 'Commodore 64 Programmer's Reference Guide' if you are interested in how the machine functions to a greater degree.

Disclaimer:

Although Breden's BASIC has been thoroughly tested and is believed to be free of mistakes, no claim is made regarding the accuracy or suitability of this software package. VISIONS (Software Factory) Limited (and its distributors) cannot assume liability or responsibility for any loss or damage arising through usage of Breden's BASIC

Errata Request:

Though, to the best of our knowledge, this software and manual are believed to be error-free, you may discover some that we have not found. In this case it would be beneficial, both to us and to future or existing owners, to be aware.

MOBSLOT

Type:	Graphics command
Format:	MOBSLOT <mob slot>,[shape string]
Example:	MOBSLOT 5,a$
Action:	The shape defined by the variable 'a$' is put into slot 5.
Parameters:	<mob slot> AEXP : 1 to 16 [shape string] SEXP
Explanation:	The MOBSLOT command is used to store shapes for sprites. The actual shape of the sprite is defined by a string variable (such as 'a$' in the example above). A sprite shape can be placed into any of the sixteen slots available, using the MOBSLOT command. Refer to the appendices for an example of how to create a sprite.

Related KEYWORDS: **SELECT, SETMOB**

EMOB

Type:	Graphics command
Format:	EMOB <mob number>
Example:	EMOB 1
Action:	Sprite number 1 will be enabled.
Parameters:	<mob slot> AEXP : 1 to 8
Explanation:	The EMOB command is used to enable, or turn on a sprite. Note that this does not necessarily mean that the sprite will be visible. The sprite could be off the screen, or could be the same colour as the screen and thus might not be immediately visible.

Related KEYWORDS: **DMOB, SETMOB**

DMOB

Type:	Graphics command
Format:	DMOB <mob number>
Example:	DMOB 1
Action:	Sprite number 1 will be disabled.
Parameters:	<mob number> AEXP : 1 to 8
Explanation:	The DMOB command is used to disable, or turn off a sprite. Thus if there is a sprite on the screen, you may turn it off at any time with this command.

Related KEYWORDS: **EMOB, SETMOB**

SELECT

Type:	Graphics command
Format:	SELECT <mob number>,<slot number>
Example:	SELECT 4,7
Action:	Sprite number 4 will look in slot 7 for its shape.
Parameters:	<mob number> AEXP : 1 to 8 <slot number> AEXP : 1 to 16
Explanation:	The SELECT command is used to define which slot a sprite gets its shape from. When a few shapes have been defined and put into different slots, simply by changing the slot number of a sprite with the SELECT command, a sprite can be animated.

Related KEYWORDS: **MOBSLOT, SETMOB**

PMOB

Type:	Graphics command
Format:	PMOB <mob number>,<priority : behind/in front>
Example:	PMOB 2,1
Action:	Sprite number 2 will go behind information on the screen.
Parameters:	<mob number> AEXP : 1 to 8 <priority : behind/in front> AEXP : 1 or 0
Explanation:	The PMOB command is used to make a sprite go behind or in front of information (characters for example) on the screen. Really clever 3 dimensional graphic effects may be achieved by having some sprites appear in front of screen information and some behind.

Related KEYWORDS: **SETMOB**

XMOB

Type:	Graphics command
Format:	XMOB <mob number>,<x expansion : on/off>, <y expansion : on/off>
Example:	XMOB 1,1,0
Action:	Sprite number 1 will be expanded horizontally (x), but will not be expanded vertically (y).
Parameters:	<mob number> AEXP : 1 to 8 <x expansion : on/off> AEXP : 1 or 0 <y expansion : on/off> AEXP : 1 or 0
Explanation:	The XMOB command is used to expand a sprite. This allows a sprite to double its size horizontally and/or vertically. This means that with both x and y expansion on, a sprite will appear four times larger than the normal size.

Related KEYWORDS: **SETMOB**

MOBCOL

Type: Graphics command

Format: MOBCOL <mob number>,<mode : multicolour/ hires>,<colour 1>,<colour 2>,<colour 3>

Example: MOBCOL 3,0,5,5,5

Action: Sprite number 3 will be a hires sprite and green (5).

Example: MOBCOL 8,1,2,1,6

Action: Sprite number 8 will be a multicolour sprite and will be red (2), white (1) and blue (6).

Parameters:

<mob number>	AEXP : 1 to 8
<mode : multicolour/hires>	AEXP : 1 or 0
<colour 1>	AEXP : 0 to 15
<colour 2>	AEXP : 0 to 15
<colour 3>	AEXP : 0 to 15

Explanation: The MOBCOL command is used to set the colour(s) of sprites. Hires sprites can only have one colour, but multicolour sprites can have three colours. Please note that even though hires sprites can only have one colour, three colours MUST be given. This is so that multicolour sprites may share the same command.

Related KEYWORDS: **SETMOB**

SETMOB

Type: Graphics command

Format: SETMOB <mob number>,<mob slot>,<priority>,
<x expansion>, <y expansion>,<mode>,
<colour 1>,<colour 2>,<colour 3>,<enable/disable>

Example: SETMOB 7,2,0,1,1,1,2,1,6,1

Action: Sprite 7 gets its shape from slot 2. It is in front of screen information. It is expanded both horizontally and vertically. It is a MULTICOLOUR SPRITE and it is coloured red, white and blue. Finally, it is ENABLED (turned on).

Parameters:

<mob number>	AEXP : 1 to 8
<mob slot>	AEXP : 1 to 16
<priority>	AEXP : 1 or 0
<x expansion>	AEXP : 1 or 0
<y expansion>	AEXP : 1 or 0
<mode>	AEXP : 1 or 0
<colour 1>	AEXP : 0 to 15
<colour 2>	AEXP : 0 to 15
<colour 3>	AEXP : 0 to 15
<enable/disable>	AEXP : 1 or 0

Explanation: The SETMOB command is perhaps the most powerful of all BREDEN's BASIC commands, with ten parameters to set. This means that in one statement it is possible to define most of a SPRITE's criteria. For a detailed explanation of each of the parameters, refer to the related SPRITE commands. In essence, the SETMOB command brings together all the main SPRITE commands. In order, they are SELECT, PMOB, XMOB, MOBCOL, DMOB and EMOB.

Related KEYWORDS: **DMOB, EMOB, MOBCOL, MOBSLOT, PMOB, SELECT, XMOB**

MOVEMOB

Type:	Graphics command
Format:	MOVEMOB <mob number>,<x position>,<y position>
Example:	MOVEMOB 4,200,100
Action:	Sprite 4 is moved to a position 200 across, 100 down.

Parameters:

<mob number>	AEXP : 1 to 8
<x position>	AEXP : 0 to 344
<y position>	AEXP : 0 to 250

Explanation: The MOVEMOB command is used to move a sprite to a position defined by <x position>,<y position>. Sprites can be made to move in straight or curved lines by means of simple FOR . . . NEXT or REPEAT . . . UNTIL loops.

Related KEYWORDS: **MMCOLL, MDCOLL**

MDCOLL

Type:	Graphics function
Format:	MDCOLL (<mob number>)
Example:	a=MDCOLL(4)
Action:	a=0 if sprite 4 has not collided with screen information. a=1 if sprite 4 has collided with screen information.
Example:	IF MDCOLL(6)=1 THEN: BCOL IRAN (1,16)
Action:	If sprite 6 has hit any screen information, the screen border will change to a random colour.
Parameters:	<mob number> AEXP : 1 to 8
Explanation:	The MDCOLL command is used to detect collisions between sprites and screen information. The command will give a value of zero if a sprite has not hit any screen information and a value of one if a sprite has hit screen information.

Related KEYWORDS: **MMCOLL**

MMCOLL

Type:	Graphics function
Format:	MMCOLL(<mob number>) or, MMCOLL(<mob number>,<mob number>)
Example:	a=MMCOLL(1)
Action:	a=0 if sprite 1 has not collided with another sprite. a=1 if sprite 1 has collided with ANY other sprite.
Example:	IF MMCOLL (3,8)=1 THEN PRINT"Sprite 3 hit sprite 8"
Action:	If sprite 3 has hit sprite 8, the computer will print 'Sprite 3 hit sprite 8' on the screen.
Parameters:	<mob number> AEXP : 1 to 8
Explanation:	The MMCOLL command is used to detect collisions between sprites. The command will give a value of zero if a sprite has not hit any other sprite and a value of one if a sprite has hit another sprite.

Related KEYWORDS: **MDCOLL**

GRAPHICS NOTE

In the following pages you will see that the range for the colour parameter is between 0 and 15. This is true whilst in hires graphics mode (HIRES). In this mode you have only one 'paint brush' to draw with.
In multicolour graphics mode (MULTI), however, you have three paint brushes. To tell the computer which brush you are using, simply add the paint brush code. Please note that this only needs to be done in multicolour graphics mode when two or more different colours are to be placed near to each other. For the paint brush codes, see the table below.

Paint brush number	Paint brush code
0	0
1	64
2	128

Examples:

PLOT 100,100,6+64
This plots a blue (6) dot with paint brush 1.

REC 100,100,10,10,7+128
This draws a yellow (7) rectangle with paint brush number 2.

HIRES

Type:	Graphics command
Format:	HIRES <border colour>,<screen colour>
Example:	HIRES 6,15
Action:	This enters hires graphics mode, gives a blue border and a white screen.
Parameters:	<border colour> AEXP : 0 to 15 <screen colour> AEXP : 0 to 15
Explanation:	The HIRES command is used to enter hires graphics mode. It sets the border and screen colours to those specified. Once this has been done, commands such as PLOT, DRAW, CLG etc., may be used.

Related KEYWORDS: **MULTI, TEXT, GRAPHICS**

MULTI

Type:	Graphics command
Format:	MULTI <border colour>,<screen colour>
Example:	MULTI 6,15
Action:	This enters multicolour graphics mode, gives a blue border and a white screen.
Parameters:	<border colour> AEXP : 0 to 15 <screen colour> AEXP : 0 to 15
Explanation:	The MULTI command is used to enter multicolour graphics mode. It sets the border and screen colours to those specified. Once this has been done, commands such as PLOT, DRAW, CLG etc., may be used.

Related KEYWORDS: **HIRES, TEXT, GRAPHICS**

CLG

Type:	Graphics command
Format:	CLG
Example:	CLG
Action:	This clears the hires and multicolour graphics screen.
Parameters:	None.
Explanation:	The CLG command is used to clear the hires and multicolour graphics screen. This is done when you want to start work on a new graphics screen.

Related KEYWORDS: **None.**

TEXT

Type:	Graphics command
Format:	TEXT
Example:	TEXT
Action:	This enters normal text mode.
Parameters:	None.
Explanation:	The TEXT command is used to leave either the hires or multicolour graphics screens. This is done when you want to edit a program or save a program etc. Note that the colours will also be restored to their previous settings. Also note that the picture and colours of the picture drawn whilst in either the hires or multicolour graphics modes will still be stored within the computer. To re-display the picture drawn see the GRAPHICS command.

Related KEYWORDS: **MULTI, HIRES, GRAPHICS**

GRAPHICS

Type:	Graphics command
Format:	GRAPHICS
Example:	GRAPHICS
Action:	This enters hires or multicolour graphics mode retaining previous colour settings for border, screen etc.
Parameters:	None.
Explanation:	The GRAPHICS command is used to re-enter the graphics screen. This command 'remembers' all the settings used previously. Say, for example, you previously used a multicolour graphics screen with a black border and black screen. When the GRAPHICS command is used from text mode, the multicolour graphics screen with black border and screen along with any pictures etc. will reappear. This command, along with the TEXT command, allows screen flipping between graphics and text screens.

Related KEYWORDS: **TEXT, HIRES, MULTI**

PLOT

Type:	Graphics command
Format:	PLOT <x position>,<y position>,<colour>
Example:	PLOT 160,100,0
Action:	This plots a black dot at position 160 across, 100 down.
Parameters:	<x position> AEXP : 0 to 319 <y position> AEXP : 0 to 199 <colour> AEXP : 0 to 15
Explanation:	The PLOT command is used to place individual dots on a graphics screen. The x and y coordinates specify the position of the dot to be plotted relative to the origin (at position 0,0) which is at the top left of the screen.

Related KEYWORDS: **DRAW, LINE, REC, CLG, HIRES, MULTI, GRAPHICS**

DRAW

Type:	Graphics command
Format:	DRAW <x position>,<y position>,<colour>
Example:	DRAW 160,100,0
Action:	This draws a line from the last point drawn or plotted at to a position 160 across, 100 down.

Parameters:

<x position>	AEXP : 0 to 319
<y position>	AEXP : 0 to 199
<colour>	AEXP : 0 to 15

Explanation: The DRAW command is used to draw a line from the last point drawn to or plotted at to the point specified by the coordinates given. The line will be drawn in the colour specified.

Related KEYWORDS: **PLOT, LINE, REC, CLG, HIRES, MULTI, GRAPHICS**

LINE

Type: Graphics command

Format: LINE <x position1>,<y position1>,<x position2>,<y position2>,<colour>

Example: LINE 0,0,160,100,6

Action: This draws a line from (0,0) to (160,100) in blue.

Parameters:

<x position1>	AEXP : 0 to 319
<y position1>	AEXP : 0 to 199
<x position2>	AEXP : 0 to 319
<y position2>	AEXP : 0 to 199
<colour>	AEXP : 0 to 15

Explanation: The LINE command is used to draw a line from the first set of coordinates given, to the second set of coordinates given in the colour specified.

Related KEYWORDS: **PLOT, DRAW, REC, CLG, HIRES, MULTI, GRAPHICS**

REC

Type:	Graphics command
Format:	REC <x position>,<y position>,<width>,<height>,<colour>
Example:	REC 100,50,30,10,5
Action:	This draws a green rectangle of width 30 units and height 10 units from position (100,50).

Parameters:

<x position>	AEXP : 0 to 319
<y position>	AEXP : 0 to 199
<width>	AEXP : 0 to 319
<height>	AEXP : 0 to 199
<colour>	AEXP : 0 to 15

Explanation: The REC command is used to draw rectangles on a graphics screen. The point specified by the coordinates represents the top left corner of the rectangle; the width and height being relative (rightwards and downwards) to the top left corner. Please note that by giving a suitable width and height it is possible to create a square.

Related KEYWORDS: **PLOT, DRAW, LINE, CLG, HIRES, MULTI, GRAPHICS**

BCOL

Type:	Graphics command
Format:	BCOL <border colour>
Example:	BCOL 5
Action:	This gives a green border colour.
Parameters:	<border colour> AEXP : 0 to 15
Explanation:	The BCOL command is used to change the colour of the screen border.

Related KEYWORDS: **DCOL, SCOL, TCOL, COLOUR**

SCOL

Type:	Graphics command
Format:	SCOL <screen colour>
Example:	SCOL 0
Action:	This gives a black screen colour.
Parameters:	<screen colour> AEXP : 0 to 15
Explanation:	The SCOL command is used to change the colour of the screen.

Related KEYWORDS: **BCOL, DCOL, TCOL, COLOUR**

	TCOL
Type:	Graphics command
Format:	TCOL <text printing colour>
Example:	TCOL 14
Action:	Characters PRINTed will now be light blue.
Parameters:	<text printing colour> AEXP : 0 to 15
Explanation:	The TCOL command is used to change the colour of text or characters to be PRINTed to the screen.

Related KEYWORDS: **BCOL, DCOL, SCOL, COLOUR**

COLOUR

Type: Graphics command

Format: COLOUR <border colour>,<screen colour>,<text colour>

Example: COLOUR 6,15,11

Action: This sets a dark blue border, white screen, charcoal colour text and clears the screen.

Parameters:

<border colour>	AEXP : 0 to 15
<screen colour>	AEXP : 0 to 15
<text colour>	AEXP : 0 to 15

Explanation: The COLOUR command is used to set the border, screen and text colours. It also clears the screen. In effect, the COLOUR command integrates the BCOL, SCOL and TCOL and CLS commands. The BCOL, SCOL and TCOL commands are useful for altering individual colour criteria without the need to specify irrelevant parameters.

Related KEYWORDS: **BCOL, SCOL, TCOL, DCOL**

DCOL

Type:	Graphics command
Format:	DCOL <character colour>
Example:	DCOL 2
Action:	All characters on the screen will turn red.
Parameters:	<character colour> AEXP : 0 to 15
Explanation:	The DCOL command is used to change the colour of all text or characters already on the screen. For those interested, the DCOL command fills colour memory ($D800 — $DBFF).

Related KEYWORDS: **BCOL, SCOL, TCOL, COLOUR**

LCASE

Type:	Graphics command
Format:	LCASE
Example:	LCASE
Action:	Characters printed/to be printed appear in lower case.
Parameters:	None.
Explanation:	The LCASE command is used to make characters already printed, or to be printed, appear in lower case.

Related KEYWORDS: **UCASE**

	UCASE
Type:	Graphics command
Format:	UCASE
Example:	UCASE
Action:	Characters printed/to be printed appear in upper case.
Parameters:	None.
Explanation:	The UCASE command is used to make characters already printed, or to be printed, appear in upper case.

Related KEYWORDS: **LCASE**

CLS

Type:	Graphics command
Format:	CLS
Example:	CLS
Action:	The text screen is cleared.
Parameters:	None.
Explanation:	The CLS command is used to clear the text screen. It replaces the messy and illegible cursor control character.

Related KEYWORDS: **None.**

CAT

Type:	Graphics command
Format:	CAT <x position>,<y position>
Example:	CAT 5,10:PRINT "Hello there !"
Action:	The cursor moves 5 places across, 10 places down and prints 'Hello there !' on the screen.
Parameters:	<x position> AEXP : 0 to 39 <y position> AEXP : 0 to 24
Explanation:	The CAT command is used to place the cursor at a place X squares across and Y places down from the top left hand corner of the screen. This is done to make the computer PRINT or INPUT at that position on the screen. This command helps make the messy cursor control characters become obsolete, thus enhancing the readability of your programs. The position (0,0) is at the top left hand corner of the screen.

Related KEYWORDS: **PRINT, INPUT**

GMODE

Type:	Graphics command
Format:	GMODE <mode number>
Example:	GMODE 2
Action:	Extended background colour mode is turned on.
Parameters:	<mode number> AEXP : 0 to 2 0: Text mode (normal) 1 : Multicolour mode 2 : Extended background colour mode
Explanation:	The GMODE command is used to select one of the three modes which are given above. Mode 0 is the one which is normally used for typing.

Related KEYWORDS: **TCOL, MCOLS, ECOLS**

MCOLS

Type:	Graphics command
Format:	MCOLS <colour 1>,<colour 2>
Example:	MCOLS 3,6
Action:	This sets light and dark blue as colours 1 and 2 respectively.
Parameters:	<colour 1> AEXP : 0 to 15 <colour 2> AEXP : 0 to 15
Explanation:	The MCOLS command is used to set two out of the four colours used in multicolour text mode. See the appendices for details on how these colours are used.

Related KEYWORDS: **SCOL, DCOL, GMODE**

ECOLS

Type:	Graphics command
Format:	ECOLS <colour 1>,<colour 2>,<colour 3>
Example:	ECOLS 3,6,12
Action:	This sets light blue, dark blue and grey as colours 1, 2 and 3 respectively.

Parameters:

<colour 1>	AEXP : 0 to 15
<colour 2>	AEXP : 0 to 15
<colour 3>	AEXP : 0 to 15

Explanation: The ECOLS command is used to set three out of the four colours used in extended background colour mode. See the appendices for details on how these colours are used.

Related KEYWORDS: **SCOL, DCOL, GMODE**

FLICKER

Type:	Graphics command
Format:	FLICKER <switch : on/off>
Example:	FLICKER 0
Action:	Most of the screen flicker related to graphics commands should disappear.
Parameters:	<switch : on/off> AEXP : 1 or 0
Explanation:	The FLICKER command is used to switch on or off screen flicker. The advantage of turning flicker off is that nice flicker-free graphics are possible. The disadvantage is that graphics are slower (about half normal speed).

Related KEYWORDS: **None.**

UDG

Type:	Graphics command
Format:	UDG
Example:	UDG
Action:	Enters UDG mode and allows use of UDG characters.
Parameters:	None.
Explanation:	The UDG command is used to enter user defined graphics mode where you may define your own characters within an 8 by 8 pixel matrix. See the SETUDG command for details on this. Please note that you should not use the graphics screen whilst in UDG mode (use CBM first) otherwise the character set will be corrupted (user defined graphics and high resolution graphics use conflicting memory).

Related KEYWORDS: **CBM, SETUDG, SUDG**

SETUDG

Type:	Graphics command
Format:	SETUDG <poke code>,[definition string]
Example:	SETUDG 0,b$
Action:	The UDG defined by the variable 'b$' is assigned to the '@' character.
Parameters:	<poke code> AEXP : 0 to 255 [definition string] SEXP
Explanation:	The SETUDG command is used to assign your own user defined character in place of a character already defined, such as the '@' character as in the example above. For detailed information on how to create your own characters please refer to the appendices.

Related KEYWORDS: **CBM, UDG, SUDG**

CBM

Type:	Graphics command
Format:	CBM
Example:	CBM
Action:	Leaves UDG mode and allows use of normal cbm characters.
Parameters:	None.
Explanation:	The CBM command is used to leave user defined graphics mode and re-allow the use of normal Commodore characters. After this command has been issued, user defined graphic symbols will not appear on the text screen.

Related KEYWORDS: **UDG, SETUDG, SUDG**

SUDG

Type:	Graphics command
Format:	SUDG [file name],<device number>
Example:	SUDG "compufont",8
Action:	This stores your own character set on disk with a name of 'compufont'.
Parameters:	[file name] SEXP : up to 16 characters <device number> AEXP
Explanation:	The SUDG command is used to store your own character set or user defined graphics (UDGs) onto a storage medium such as tape or disk. This means that at a later date it is possible to LOAD back in the character set from tape or disk. See the appendix on user defined graphics for details.

Related KEYWORDS: **UDG, MEMLOAD, COPY, CBM**

SHIRES

Type:	Graphics command
Format:	SHIRES [file name],<device number>
Example:	SHIRES "spiral",8
Action:	This stores a picture drawn on the hires or multicolour graphics screen.
Parameters:	[file name] SEXP : up to 15 characters <device number> AEXP
Explanation:	The SHIRES command is used to store your own pictures drawn on the hires or multicolour graphics screen onto a storage medium such as tape or disk. This means that at a later date it is possible to LOAD back in the picture from tape or disk. See the appendix on user high resolution graphics for details.

Related KEYWORDS: **MEMLOAD, COPY, GRAPHICS**

COPY

Type:	Graphics command
Format:	COPY
Example:	COPY
Action:	This copies either (1) the UDGs up in memory, or (2) the second picture file up in memory.
Parameters:	None.
Explanation:	The COPY command is used to copy either (1) the UDGs up in memory, or (2) the second picture file up in memory. This needs to be done after LOADing UDG character sets or picture files from tape or disk. See the appendices on user defined graphics or high resolution graphics for details on usage.

Related KEYWORDS: **MEMLOAD, UDG, GRAPHICS**

ALTER

Type:	Graphics command
Format:	ALTER <border colour>,<screen colour>
Example:	ALTER 6,5
Action:	This alters the store colour settings for the border and screen colours to blue (6) and green (5) for multicolour and high resolution graphics modes.
Parameters:	<border colour> AEXP : 0 to 15 <screen colour> AEXP : 0 to 15
Explanation:	For the ALTER command to have any visible effect, the computer must be in text mode when it is used. Then, to see its effect on your picture stored on the graphics screen, use the GRAPHICS command.

Related KEYWORDS: **GRAPHICS**

ISOUND

Type:	Sound command
Format:	ISOUND
Example:	ISOUND
Action:	This will initialize all the sound settings (volume etc.) within the sound synthesis chip.
Parameters:	None.
Explanation:	The ISOUND command is used to initialize the sound settings which the Commodore 64 uses to produce sound. It is recommended that this command precedes any BASIC code which relates to sound.

Related KEYWORDS: **None.**

VOLUME

Type:	Sound command
Format:	VOLUME <volume level>
Example:	VOLUME 15
Action:	This sets the volume level to 15 (maximum setting).
Parameters:	<volume level> AEXP : 0 to 15
Explanation:	The VOLUME command is used to set the volume level of all sound produced by the computer. Although there are other sound features which can control the loudness of sound, this command has control over all others which affect the loudness in any way.

Related KEYWORDS: **None.**

ENVELOPE

Type: Sound command

Format: ENVELOPE <voice number>,<attack>,<decay>, <sustain>,<release>

Example: ENVELOPE 1,0,0,15,9

Action: This sets an attack of 0, decay of 0, sustain of 15, and a release of 9 on voice 1.

Parameters:

<voice number>	AEXP : 1 to 3
<attack>	AEXP : 0 to 15
<decay>	AEXP : 0 to 15
<sustain>	AEXP : 0 to 15
<release>	AEXP : 0 to 15

Explanation: The ENVELOPE command is used to set the criteria which control the way the sound amplitude envelope is formed. These criteria are namely, attack, decay, sustain and release. With the exception of sustain, which is a volume level, attack, decay and release are all time measurements which control how the volume is affected during the envelope cycle. Sustain is a volume level ranging from 0 (silent) to 15 (loud) in 16 linear steps. Therefore, a sustain of 7 or 8 is half volume. The following table shows the values that should be specified with the ENVELOPE command in accordance to attack, decay and release times. For those 'non-metricians' out there, 'ms' stands for milliseconds, and 's', of course, stands for seconds.

Value	Attack rate	Decay/Release rate
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Related KEYWORDS: **ATTACK, RELEASE**

WAVE

Type:	Sound command
Format:	WAVE <voice number>,<waveform type>
Example:	WAVE 2,1
Action:	This sets sawtooth waveform on voice 2.
Parameters:	<voice number> AEXP : 1 to 3 <waveform type> AEXP : 0 to 3 0 : triangle waveform 1 : sawtooth waveform 2 : pulse waveform 3 : white noise waveform
Explanation:	The WAVE command is used to set the waveform for a voice. Depending which waveform is selected, different sounds are produced. Incidentally, with pulse waveform there are 4096 different settings!

Related KEYWORDS: **None.**

PWIDTH

Type:	Sound command
Format:	PWIDTH <voice number>,<pulse width>
Example:	PWIDTH 3,2048
Action:	This sets a pulse width of 2048 on voice 3.
Parameters:	<voice number> AEXP : 1 to 3 <pulse width> AEXP : 0 to 4095
Explanation:	The PWIDTH command is used to set the pulse width when using pulse waveform on a voice. There are a variety of tones that may be produced with different pulse widths, so experiment!

Related KEYWORDS: **WAVE**

ATTACK

Type:	Sound command
Format:	ATTACK <voice number>
Example:	ATTACK 1
Action:	This starts the attack/decay/sustain cycle on voice 1.
Parameters:	<voice number> AEXP : 1 to 3
Explanation:	The ATTACK command is used to initiate the attack/decay/sustain cycle of the envelope. This command must be used, or have been used, in order for sounds produced to be audible. See the appendix on sound for details.

Related KEYWORDS: **ENVELOPE, RELEASE**

RELEASE

Type:	Sound command
Format:	RELEASE <voice number>
Example:	RELEASE 2
Action:	This starts the release cycle on voice 2.
Parameters:	<voice number> AEXP : 1 to 3
Explanation:	The RELEASE command is used to initiate the release cycle of the envelope. See the appendix on sound for details.

Related KEYWORDS: **ENVELOPE, ATTACK**

PITCH

Type:	Sound command
Format:	PITCH <voice number>,<voice frequency>
Example:	PITCH 1,5000
Action:	This sets a frequency of 5000 on voice 1.
Parameters:	<voice number> AEXP : 1 to 3 <voice frequency> AEXP : 0 to 65535
Explanation:	The PITCH command is used to set the frequency or pitch on a voice. There is a vast range of frequencies that may be chosen. A low frequency gives a low, rumbling sound, whereas a high frequency gives a high, piercing sound. Sweeping through a range of frequencies, either from high to low frequencies or vice versa, can produce pleasing effects, especially when varying the size of the sweeping step. Filtering and other effects such as synchronization or ring modulation combined with use of the PITCH command is very powerful. Feel free to experiment, as many sound effects are discovered through trial and error.

Related KEYWORDS: **PLAY, CHRD**

PLAY

Type: Sound command

Format: PLAY <voice number>,<octave number>,<note number>

Example: PLAY 2,4,11

Action: This plays A ♯ (B flat) from the fourth octave on voice 2.

Parameters:

<voice number>	AEXP : 1 to 3
<octave number>	AEXP : 1 to 8
<note number>	AEXP : 1 to 12

Explanation: The PLAY command is used to select a specific note from a particular octave. The voice, octave and note to be specified with the PLAY command could come from DATA statements, sequential files on tape or disk, or even from memory. This makes the PLAY command versatile in its usage. PLAY, combined perhaps with CHRD, makes it relatively simple to play tunes from sheet music. The notes must, however, initially be converted into voice, octave and note form. The note values are derived from the table below.

Note number	Corresponding note
1	C
2	C ♯ (D flat)
3	D
4	D ♯ (E flat)
5	E
6	F
7	F ♯ (G flat)
8	G
9	G ♯ (A flat)
10	A
11	A ♯ (B flat)
12	B

Related KEYWORDS: **CHRD, PITCH**

CHRD

Type: Sound command

Format: CHRD <voice number>,<octave number>,<note number>,<voice number>,<octave number>,<note number>,.....

Example: CHRD 1,4,1,2,4,5

Action: This plays a chord of C and E from octave 4 on voices 1 and 2.

Example: CHRD 1,4,1,2,4,5,3,4,10

Action: This plays a chord of C, E and G from octave 4 on voices 1, 2 and 3.

Parameters:

<voice number>	AEXP : 1 to 3
<octave number>	AEXP : 1 to 8
<note number>	AEXP : 1 to 12

Explanation: The CHRD command is used to play chords. Note that playing a chord involves at least two voices. Thus, chords involving two or three voices are possible. The numbers representing the voice, octave and note number could come from DATA statements, sequential files on tape or disk, or even from memory. This makes the CHRD command versatile in its usage. CHRD, combined with PLAY, makes it relatively simple to play tunes from sheet music. The notes must, however, initially be converted into voice, octave and note form. The note values are derived from the table below.

Note number	Corresponding note
1	C
2	C ♯ (D flat)
3	D
4	D ♯ (E flat)
5	E
6	F
7	F ♯ (G flat)
8	G
9	G ♯ (A flat)
10	A
11	A ♯ (B flat)
12	B

Related KEYWORDS: **PLAY, PITCH**

SYNC

Type: Sound command

Format: SYNC <variable frequency voice>,<switch : on/off>

Example: SYNC 3,1

Action: This gives synchronization between voice 3 and voice 2.

Parameters:

<variable frequency voice>	AEXP : 1 to 3
<switch : on/off>	AEXP : 1 or 0

Explanation: The SYNC command is used to synchronize the fundamental frequencies of two voices. By playing a note of arbitrary frequency on the 'constant frequency voice' (ideally of lower frequency than the frequency on the 'variable frequency voice') and then sweeping through a range of frequencies on the 'variable frequency voice', a wide variety of complex harmonic sounds can be produced. In accordance with the following table, the voice chosen for the variable frequency always determines the voice that MUST be used for the constant frequency.

Variable frequency voice	Constant frequency voice
1	3
2	1
3	2

Related KEYWORDS: **None.**

RMOD

Type: Sound command

Format: RMOD <variable frequency voice>,<switch : on/off>

Example: RMOD 1,1

Action: This gives ring modulation between voice 1 and voice 3.

Parameters:

<variable frequency voice>	AEXP : 1 to 3
<switch : on/off>	AEXP : 1 or 0

Explanation: The RMOD command is used to ring modulate the frequencies of two voices. The effect is similar to SYNC (see SYNC) except that both frequency and volume can be varied simultaneously. Ring modulation replaces the triangle waveform output of the variable frequency voice with the ring modulated combination of the frequencies of the variable and constant frequency voices. By playing a note of arbitrary frequency on the 'constant frequency voice' and then sweeping through a range of frequencies on the 'variable frequency voice', a wide variety of non harmonic overtone structures can be produced. Try using the opposite order for the voice frequencies, and try varying both the frequencies at the same time; it sounds amazing! In accordance with the following table, the voice chosen for the variable frequency always determines the voice that must be used for the constant frequency. Please note that the waveform of the variable frequency voice MUST be set to triangle for ring modulation to work.

Variable frequency voice	Constant frequency voice
1	3
2	1
3	2

Related KEYWORDS: **None.**

CHN3

Type:	Sound command
Format:	CHN3 <switch : on/off>
Example:	CHN3 0
Action:	This prevents any undesirable sound from voice 3.
Parameters:	<switch : on/off> AEXP : 1 or 0
Explanation:	The CHN3 command is used to connect or disconnect the audible output of voice 3 from the direct audio path. When using voice 3 for special modulation purposes it is usual to disconnect voice 3 from the direct audio path. This is done by using a value of 0 with the CHN3 command (i.e. 'CHN3 0').

Related KEYWORDS: **OSC3, ENV3**

OSC3

Type:	Sound function
Format:	OSC3
Example:	a=OSC3
Action:	a=a value depending on the status of voice 3.
Parameters:	None.
Explanation:	The OSC3 function is used to return a value representing the high byte of oscillator 3. The character of the sequence of values returned is dependent upon the waveform selected for voice 3.
Triangle waveform	The sequence of values returned will increment from 0 to 255, then will decrement down to 0. This cycle repeats.
Sawtooth waveform	The sequence of values returned will increment from 0 to 255, then leap to 0. This cycle repeats.
Pulse waveform	The sequence of values returned will alternate between 0 and 255.
Noise waveform	The sequence of values returned will be random. This is useful for a random number generator. The numbers will be between 0 and 255.
	There are many possible applications for this function, although it is likely to be used mainly as a modulation generator for sound effects. The value returned may be added to or subtracted from the filter cutoff frequency for example, or used to define the pulse width in real time. Vibrato can be produced by setting the waveform of voice 3 to triangle, setting the frequency of voice 3 to about 7 Hz (using the PITCH command), and using the OSC3 value (after appropriate scaling) to the frequency of a different voice. Normally, when voice 3 is being used for this purpose, the sound output of voice 3 should be turned off (by using 'CHN3 0').

Related KEYWORDS: **CHN3, ENV3**

ENV3

Type:	Sound function
Format:	ENV3
Example:	a=ENV3
Action:	a=a value depending on the status of the envelope generator on voice 3.
Parameters:	None.
Explanation:	The ENV3 function is used to return a value representing the changing state of the envelope generator on voice 3. To use this function properly the ATTACK or RELEASE commands must have been used just before the ENV3 function is used. As with the OSC3 function, the value returned by this function may be added to the filter cutoff frequency or frequency of another voice etc.

Related KEYWORDS: **CHN3, OSC3**

DVOICE

Type:	Sound command
Format:	DVOICE <voice number>
Example:	DVOICE 1
Action:	This disables voice 1.
Parameters:	<voice number> AEXP : 1 to 3
Explanation:	The DVOICE command is used to turn off a particular voice. The interesting thing is that when the converse command, EVOICE, is used, the last note or frequency played on that voice is restored. The combination of the DVOICE and EVOICE commands can be used to create a rhythm effect. The speed of the rhythm is controlled by the speed a voice is enabled and disabled with the EVOICE and DVOICE commands.

Related KEYWORDS: **EVOICE**

EVOICE

Type:	Sound command
Format:	EVOICE <voice number>
Example:	EVOICE 2
Action:	This enables voice 2.
Parameters:	<voice number> AEXP : 1 to 3
Explanation:	The EVOICE command is used to turn on a particular voice. The interesting thing is that when the converse command, DVOICE, is used, the last note or frequency played on that voice is 'remembered'. The combination of the DVOICE and EVOICE commands can be used to create a rhythm effect. The speed of the rhythm is controlled by the speed a voice is enabled and disabled with the EVOICE and DVOICE commands.

Related KEYWORDS: **DVOICE**

FVOICE

Type:	Sound command
Format:	FVOICE <voice number>,<select/deselect>
Example:	FVOICE 3,1
Action:	This passes all output from voice 3 through the filter.
Parameters:	<voice number> AEXP : 1 to 3 <select/deselect> AEXP : 1 or 0
Explanation:	The FVOICE command is used to select or deselect filtering of particular voices. As in the example above, if a value of one is used, that voice will be passed through the filter. If, however, a value of zero is used, that voice will not be passed through the filter. This command MUST be used if you intend to filter a voice.

Related KEYWORDS: **FTYPE, RES, COFF**

FTYPE

Type: Sound command

Format: FTYPE <filter type>

Example: FTYPE 1

Action: This gives low pass filtering.

Parameters: <filter type> AEXP : 1 to 7

Explanation: The FTYPE command is used to select various types and mixtures of sound filtering. Three main types of filtering are provided for, although it is possible to achieve a maximum of seven combinations of these three. The three main types are BAND PASS, LOW PASS and HIGH PASS filtering. Brief details are given below.

LOW PASS

All frequency components ABOVE the filter cutoff frequency are attenuated at a rate of 12 decibels per octave. LOW PASS filtering produces full bodied sounds.

BAND PASS

All frequency components ABOVE and BELOW the filter cutoff frequency are attenuated at a rate of 6 decibels per octave. BAND PASS filtering produces thin, open sounds.

HIGH PASS

All frequency components BELOW the filter cutoff frequency are attenuated at a rate of 12 decibels per octave. HIGH PASS filtering produces tinny, buzzy sounds.

Filter value	Filter type
1	LOW PASS
2	BAND PASS
3	BAND and LOW PASS
4	HIGH PASS
5	HIGH and LOW PASS
6	HIGH and BAND PASS
7	HIGH, BAND and LOW PASS

Related KEYWORDS: **FVOICE, RES, COFF, FXINP**

RES

Type:	Sound command
Format:	RES <level of filter resonance>
Example:	RES 15
Action:	This selects the maximum amount of filter resonance.
Parameters:	<level of filter resonance> AEXP : 0 to 15
Explanation:	The RES command is used to select the amount of resonance for the sound filtering process. Resonance is a peaking effect which emphasises frequency components at the cutoff frequency of the filter, creating a more emphatic sound. There are 16 settings for resonance, ranging linearly from 0 (no resonance) to 15 (maximum resonance).

Related KEYWORDS: **FTYPE, FVOICE, COFF, FXINP**

COFF

Type:	Sound command
Format:	COFF <filter cutoff frequency>
Example:	COFF 1024
Action:	This sets the filter cutoff frequency to 1024.
Parameters:	<filter cutoff frequency> AEXP : 0 to 2047
Explanation:	The COFF command is used to set the cutoff frequency for the filter. This command, coupled with the FTYPE command, determines how much of the frequency components are filtered. Of course, for this to have any effect, the voice(s) in question must have been selected to pass through the filter (See FVOICE). The filter cutoff frequency ranges linearly from 0 to 2047. Experiment!

Related KEYWORDS: **FTYPE, FVOICE, RES, FXINP**

FXINP

Type:	Sound command
Format:	FXINP <switch : on/off>
Example:	FXINP 1
Action:	This allows filtering of external input sources.
Parameters:	<switch : on/off> AEXP : 1 to 0
Explanation:	The FXINP command is used to turn on/off the feature to filter external inputs (a tape recorder signal, for example). The signal should be connected to pin 5 (audio in) on the audio/video port of the computer. If you need further details please refer to the Programmer's Reference Guide (page 472-473). Once the signal has been processed by the filter it will be passed to pin 3 (audio out) of the audio/video port. From here it may be passed into an amplifier for output, or you can use the filtered signal for your own applications.

Related KEYWORDS: **FTYPE, RES, COFF**

ASKEY

Type:	I/O function
Format:	ASKEY
Example:	k=ASKEY
Action:	k=the ascii value of the key pressed down on the keyboard.
Parameters:	None.
Explanation:	The ASKEY function is used to determine the ascii value of the key pressed down on the keyboard. If no key is pressed, a value of zero will be returned. Please note that unless you have pressed a key that repeats (cursor key for example), the ascii value of the key will be returned once only, until a key is pressed again.

Related KEYWORDS: **FKEY, CKEY, KCODE, INKEY$**

CKEY

Type: I/O function

Format: CKEY

Example: c=CKEY

Action: c=a value representing which control key is pressed.

Parameters: None.

Explanation: The CKEY function is used to determine which (if any) control key is pressed on the keyboard. Control keys include the shift key, the Commodore logo key and the CTRL key. If no control key is pressed, a value of zero will be returned, otherwise a value (1 to 7) will be returned, depending on which key, or combination of keys, is/are pressed. The following table illustrates the values returned by the control keys.

Value returned	Control key(s) pressed
0	No control key
1	Shift key
2	Commodore logo key
3	Commodore logo and shift
4	Ctrl key
5	Ctrl and shift
6	Ctrl and Commodore logo key
7	Ctrl, Commodore logo and shift

Related KEYWORDS: **ASKEY, FKEY, KCODE, INKEY$**

	KCODE
Type:	I/O function
Format:	KCODE
Example:	k=KCODE
Action:	k=a value representing which key is pressed.
Parameters:	None.
Explanation:	The KCODE function is used to determine which (if any) key is pressed on the keyboard. The value returned by this function is generated by the system interrupt and is quite different to the ascii value of the same key. The main difference of this function over the ASKEY function, apart from the value returned, is that it returns values continuously after a key has been depressed. Note that if no key is pressed, a value of 64 is returned.

Related KEYWORDS: **ASKEY, FKEY, CKEY, INKEY$**

INKEY$

Type:	I/O function
Format:	INKEY$
Example:	a$=INKEY$
Action:	a$=the string representation of the key pressed.
Parameters:	None.
Explanation:	The INKEY$ function is used to determine which (if any) key is pressed on the keyboard. The string returned represents which key is pressed. For example, if the 'g' key is held down when this function is used, the string "g" is returned. This is in ascii form.

Related KEYWORDS: **ASKEY, FKEY, CKEY, KCODE**

	FKEY
Type:	I/O function
Format:	FKEY
Example:	f=FKEY
Action:	f=a value representing the function key pressed.
Parameters:	None.
Explanation:	The FKEY function is used to determine which (if any) function key is pressed on the keyboard. If no function key is pressed, a value of zero will be returned, otherwise a value (1 to 16) will be returned, depending on which key is pressed.

Related KEYWORDS: **ASKEY, CKEY, KCODE, INKEY$**

JOY

Type: I/O function

Format: JOY (<control port number>)

Example: a=JOY (2)

Action: a= the direction joystick 2 is in.

Parameters: <control port number> AEXP : 1 or 2

Explanation: The JOY function is used to determine the direction of a joystick. A whole range of values are returned depending on the direction.

Value returned	Direction of joystick
1	Stationary (normal)
2	Forward
3	Forward and right
4	Right
5	Right and down
6	Down
7	Down and left
8	Left
9	Left and forward

Related KEYWORDS: **JOYBUT**

JOYBUT

Type:	I/O function
Format:	JOYBUT (<control port number>)
Example:	a=JOYBUT (1)
Action:	a=1 if the fire button on joystick 1 was pressed.
Parameters:	<control port number> AEXP : 1 or 2
Explanation:	The JOYBUT function is used to determine whether the fire button on a joystick is pressed. If the fire button is pressed, a value of 1 is returned, otherwise 0 is returned.

Related KEYWORDS: **JOY**

PADDLE

Type:	I/O function
Format:	PADDLE (<paddle number>)
Example:	a=PADDLE (1)
Action:	a=a value representing the degree of turn on paddle 1.
Parameters:	<paddle number> AEXP : 1 to 4
Explanation:	The PADDLE function is used to determine the degree of turn induced on a particular paddle. Please note that there are only 2 control ports, but 4 paddles may be used. Paddles 1 and 2 share control port 1, and paddles 3 and 4 share control port 2.

Related KEYWORDS: **PTRIG**

PTRIG

Type:	I/O function
Format:	PTRIG (<paddle number>)
Example:	a=PTRIG (4)
Action:	a=1 if the fire button on paddle 4 is pressed.
Parameters:	<paddle number> AEXP : 1 to 4
Explanation:	The PTRIG function is used to determine whether the fire button of a particular paddle is pressed. If the fire button is pressed a value of 1 is returned, otherwise a value of 0 is returned.

Related KEYWORDS: **PADDLE**

LPENX

Type:	I/O function
Format:	LPENX
Example:	x=LPENX
Action:	x=the position of the light pen across the screen.
Parameters:	None.
Explanation:	The LPENX function is used to determine the horizontal position of a light pen on the screen. This, used in conjunction with the LPENY function, will give the position pointed at by the light pen. Knowing this, you may wish to plot a point, draw a line, select an option from a menu etc. Please note that the light pen MUST ONLY be placed in control port one (the port nearest the front of the computer and furthest away from the power switch).

Related KEYWORDS: **LPENY**

LPENY

Type:	I/O function
Format:	LPENY
Example:	y=LPENY
Action:	y=the position of the light pen down the screen.
Parameters:	None.
Explanation:	The LPENY function is used to determine the vertical position of a light pen on the screen. This, used in conjunction with the LPENX function, will give the position pointed at by the light pen. Knowing this, you may wish to plot a point, draw a line, select an option from a menu etc. Please note that the light pen MUST ONLY be placed in control port one (the port nearest the front of the computer and furthest away from the power switch).

Related KEYWORDS: **LPENX**

BREDEN'S BASIC

DISK DIRECTORY

On the disk provided there are a variety of programs. A list of them is given below. With all of the programs, except 'FUNCTION KEYS', the loading instructions are:

```
LOAD "NAME", 8
RUN
```

With the file called 'FUNCTION KEYS' the loading instructions are:

```
MEMLOAD "FUNCTION KEYS", 8
```

The list of files on the disk are:

FUNCTION KEYS
FONT.LOA
SPIRAL.LOA
POLY.BAS
GRIDSOUND.BAS
SYNC.BAS
RMOD.BAS
FILTER 1.BAS
FILTER 2.BAS
SPRITES 1.BAS
SPRITES 2.BAS
SPRITES 3.BAS
ECOLS
DEMO

DERR

Type:	Disk command
Format:	DERR
Example:	DERR
Action:	The disk error message is displayed on the screen.
Parameters:	None.
Explanation:	The DERR command is used to find out what error has occurred with the disk drive. After this command has been given, the error light on the disk drive should go off, and the error message will be displayed on the screen.

Related KEYWORDS: **DIR, DOS**

DIR

Type:	Disk command
Format:	DIR or, DIR <drive number> or, DIR [pattern match]
Example:	DIR
Action:	All files on drive 0 are displayed. (saves typing).
Example:	DIR 0
Action:	All files on drive 0 are displayed.
Example:	DIR "1:key★=p"
Action:	All programs beginning with 'key' on drive 1 are displayed.
Parameters:	<drive number> AEXP : 0 or 1 [pattern match] SEXP : pattern string
Explanation:	The DIR command is used to display all or some of the files on the disk onto the screen. The feature allowing 'pattern matching' is especially useful. Space here does not permit an explanation of all the possibilities of 'pattern matching', so please refer to your disk drive manual for details, it's useful! If, at any time you wish to pause the list of file names, simply press the SHIFT or SHIFT LOCK key. Release the SHIFT or SHIFT LOCK key to resume the listing.

Related KEYWORDS: **DERR, DOS**

DOS

Type:	Disk command
Format:	DOS [command string]
Example:	DOS "I0"
Action:	The disk in drive 0 is initialized.
Parameters:	[command string] SEXP
Explanation:	The DOS command is used to send commands to the disk drive. In essence, the DOS command replaces conventional awkward structures of: OPEN 15,8,15,"I": CLOSE 15 with: DOS "I" The example above shows only one of the many different commands that can be sent to the disk drive with this command. For more examples, please refer to the appendices.

Related KEYWORDS: **DERR, DIR**

PI

Type:	Mathematical function
Format:	PI
Example:	PRINT PI
Action:	The mathematical constant, PI is printed (3.14159265).
Parameters:	None.
Explanation:	The PI function is used in mathematical formulae, especially when relating to circles. Wherever you would normally use the pi symbol, you can now use the more readable form of PI.

Related KEYWORDS: **None.**

FRAC

Type:	Mathematical function
Format:	FRAC (<value>)
Example:	a=FRAC (PI)
Action:	a=0.141592653.
Parameters:	<value> AEXP : 0 to 32767
Explanation:	The FRAC function is used to find the fractional part of a value (the part to the right of the decimal point). Thus, the value returned is between 0 and almost 1. This command is the converse of the INT command.

Related KEYWORDS: **INT**

MOD

Type:	Mathematical function
Format:	MOD (<value>,<modulo>)
Example:	PRINT MOD (23,4)
Action:	3 is printed (23 divided by 4 is 5 remainder 3).
Parameters:	<value> AEXP <modulo> AEXP
Explanation:	The MOD function is used to find the remainder when one value is divided by another. Note that the sign (positive or negative) must be the same in the two values given.

Related KEYWORDS: **DIV**

DIV

Type:	Mathematical function
Format:	DIV (<value1>),<value2>)
Example:	PRINT DIV (23,4)
Action:	5 is printed (23 divided by 4 is 5).
Parameters:	<value1> AEXP <value2> AEXP
Explanation:	The DIV function is used to find the integer part of the answer when one value is divided by another. Note that the sign (positive or negative) must be the same in the two values given.

Related KEYWORDS: **MOD**

EXCL

Type:	Mathematical function
Format:	EXCL (<value1>,<value2>)
Example:	PRINT EXCL (128,1)
Action:	129 is printed.
Parameters:	<value1> AEXP : 0 to 65535 <value2> AEXP : 0 to 65535
Explanation:	The EXCL function is used to find the result after a value has been exclusive ORed with another. The exclusive OR function works on binary bits. It compares the two values bitwise. If both bits are different, the resulting bit in the answer will be a '1'. If both bits are '1', the resulting bit in the answer will be a '0'. If both bits are '0', the resulting bit in the answer will be a '0'.

Related KEYWORDS: **OR, AND, CBIT, FBIT, SBIT, TBIT**

♯ %

Type:	Mathematical function
Format:	♯ % <binary value>
Example:	PRINT ♯ %11000010
Action:	194 is printed.
Parameters:	<binary value> BEXP : 00000000 to 11111111
Explanation:	The ♯ % function is used to represent a value in binary notation. It can be used in expressions such as : a= ♯ %11000010 + 1★40 + c or it can be used simply to convert binary into decimal. However, if it is being used for the latter, see the NUMBER command.

Related KEYWORDS: **♯ $, NUMBER**

♯ $

Type:	Mathematical function
Format:	♯ $ <hexadecimal value>
Example:	PRINT ♯ $1000
Action:	4096 is printed.
Parameters:	<hexadecimal value> HEXP : 0000 to ffff
Explanation:	The ♯ $ function is used to represent a value in hexadecimal notation. It can be used in expressions such as : a= ♯ $D800 + 1★40 + c or it can be used simply to convert hexadecimal into decimal. However, if it is being used for the latter, see the NUMBER command.

Related KEYWORDS: ♯ **%, NUMBER**

RAD

Type:	Mathematical function
Format:	RAD (<angle in degrees>)
Example:	PRINT RAD (180)
Action:	3.14159265 is printed.
Parameters:	<angle in degrees> AEXP
Explanation:	The RAD function is used to convert degrees into radians. As most people think of angles in terms of degrees, and computers think in radians, this will help those who don't use radians.

Related KEYWORDS: **DEG**

DEG

Type:	Mathematical function
Format:	DEG (<value in radians>)
Example:	PRINT DEG (3.14159265)
Action:	180 is printed.
Parameters:	<value in radians> AEXP
Explanation:	The DEG function is used to convert radians into degrees.

Related KEYWORDS: **RAD**

LSB

Type:	Mathematical function
Format:	LSB (<value>)
Example:	PRINT LSB (56432)
Action:	112 is printed.
Parameters:	<value> AEXP : 0 to 65535
Explanation:	The LSB function is used to find the least significant byte of a 2 byte value. As a byte consists of 8 bits, the value returned by this function will always be in the range 0 to 255. This is the equivalent of using a value of 256 as the modulo with the MOD function.

Related KEYWORDS: **MSB**

MSB

Type:	Mathematical function
Format:	MSB (<value>)
Example:	PRINT MSB (56432)
Action:	220 is printed.
Parameters:	<value> AEXP : 0 to 65535
Explanation:	The MSB function is used to find the most significant byte of a 2 byte value. As a byte consists of 8 bits, the value returned by this function will always be in the range 0 to 255. This is the equivalent of using a value of 256 as the divisor with the DIV function.

Related KEYWORDS: **LSB**

TBIT

Type:	Mathematical function
Format:	TBIT (<value>,<bit number>)
Example:	PRINT TBIT (255,4)
Action:	1 is printed.
Parameters:	<value> AEXP : 0 to 255 <bit number> AEXP : 0 to 7
Explanation:	The TBIT function is used to test a specific bit within a byte. If the bit checked for is 'ON' (set) then a value of 1 is returned. However, if the bit checked for is 'OFF' (clear) then a value of 0 is returned. For example, for a value of 7, checking for bits 0,1 or 2 would cause a value of 1 to be returned, as these bits are all set.

Related KEYWORDS: **SBIT, CBIT, FBIT**

SBIT

Type:	Mathematical function
Format:	SBIT (<value>,<bit number>)
Example:	PRINT SBIT (128,1)
Action:	130 is printed.
Parameters:	<value> AEXP : 0 to 255 <bit number> AEXP : 0 to 7
Explanation:	The SBIT function is used to set a specific bit within a byte. This is functionally similar to the OR command, except that with SBIT it is immediately obvious which bit is being set.

Related KEYWORDS: **TBIT, CBIT, FBIT**

CBIT

Type:	Mathematical function
Format:	CBIT (<value>,<bit number>)
Example:	PRINT CBIT (129,0)
Action:	128 is printed.
Parameters:	<value> AEXP : 0 to 255 <bit number> AEXP : 0 to 7
Explanation:	The CBIT function is used to clear a specific bit within a byte. This is functionally similar to the AND command, except that with CBIT it is immediately obvious which bit is being cleared.

Related KEYWORDS: **TBIT, SBIT, FBIT**

FBIT

Type:	Mathematical function
Format:	FBIT (<value>,<bit number>)
Example:	PRINT FBIT (128,2)
Action:	132 is printed.
Example:	PRINT FBIT (132,2)
Action:	128 is printed.
Parameters:	<value> AEXP : 0 to 255 <bit number> AEXP : 0 to 7
Explanation:	The FBIT function is used to flip a specific bit within a byte. This is functionally similar to the EXCL command, except that with FBIT it is immediately obvious which bit is being flipped. Flipping a bit is simply a process of replacing the bit with its opposite. For example, if a bit was set (ON, '1'), flipping it would cause it to become clear (OFF, '0'), and vice versa.

Related KEYWORDS: **TBIT, SBIT, CBIT**

FRAN

Type:	Mathematical function
Format:	FRAN (<command>,<range>)
Example:	x=FRAN (1,1000)
Action:	x=a random number between 0 and 1000 (floating point).
Parameters:	<command> AEXP : –1, 0 or 1 <range> AEXP
Explanation:	The FRAN function is used to generate and return a random number between 0 and the range specified, in floating point format. The command parameter determines the way in which the random number sequence is formed. If the value is 1, the same 'pseudo random' sequence of numbers is returned, starting from a given seed value. The seed value used to generate 'random' numbers is set on power-up of the computer, but may be changed by using –1 as the command with the FRAN command. If the value of the command is 0, the random number is generated from an internal timer.

Related KEYWORDS: **IRAN**

IRAN

Type:	Mathematical function
Format:	IRAN (<command>,<range>)
Example:	x=IRAN (1,1000)
Action:	x=a random number between 0 and 1000 (integer).
Parameters:	<command> AEXP : −1, 0 or 1 <range> AEXP
Explanation:	The IRAN function is used to generate and return a random number between 0 and the range specified, in integer format. The command parameter determines the way in which the random number sequence is formed. If the value is 1, the same 'pseudo random' sequence of numbers is returned, starting from a given seed value. The seed value used to generate 'random' numbers is set on power-up of the computer, but may be changed by using −1 as the command with the IRAN command. If the value of the command is 0, the random number is generated from an internal timer.

Related KEYWORDS: **FRAN**

DEEK

Type:	Mathematical function
Format:	DEEK (<memory address>)
Example:	PRINT DEEK (43)
Action:	The start address of BASIC memory is displayed.
Parameters:	<memory address> AEXP : 0 to 65535
Explanation:	The DEEK function is used to obtain a value read from 2 addresses in memory. The low byte (least significant byte) is taken from the memory address specified, and the high byte (most significant byte) is taken from the memory address specified+1. The value is then returned to you. The converse of DEEK is DOKE. Commodore's own BASIC supports two commands similar, PEEK and POKE, but they only act on single bytes.

Related KEYWORDS: **DOKE, POKE, PEEK**

DOKE

Type:	Mathematical command
Format:	DOKE <address in memory>,<value>
Example:	DOKE 45,4096
Action:	0 (low byte) is put in location 45, 16 (high byte) is put in location 46.
Parameters:	<address in memory> AEXP : 0 to 65535 <value> AEXP : 0 to 65535
Explanation:	The DOKE command is used to put a value into a 2 byte address, in low/high byte order. This will have the obvious use of re-directing pointers/vectors to experienced BASIC and/or machine language programmers.

Related KEYWORDS: **DEEK, POKE, PEEK**

PUT

Type:	Mathematical command
Format:	PUT <memory location>,<value>
Example:	PUT 2048,66
Action:	This puts a value of 66 into memory location 2048.
Parameters:	<memory location> AEXP : 0 to 65535 <value> AEXP : 0 to 255
Explanation:	The PUT command is identical in usage to the POKE command. However, Commodore's POKE command will NOT work with all BREDEN's BASIC expressions. Thus, to overcome this problem, use the PUT command in place of the POKE command and all expressions will work.

Related KEYWORDS: **POKE, PEEK, DOKE, DEEK**

NUMBER

Type:	Mathematical command
Format:	NUMBER <value>
Example:	NUMBER 32768
Action:	The three conversions will be printed on the screen.
Parameters:	<value> AEXP : 0 to 65535
Explanation:	The NUMBER command is used to convert a value in one numeric base (decimal for example) into all three numeric bases supported by BREDEN's BASIC. The three numeric bases are binary (base 2), decimal (base 10) and hexadecimal (base 16). The conversions will be printed on the screen and in ascending order of base, binary first. With the binary conversion, the first eight bits (high byte) will be highlighted to help prevent possible confusion between bytes.

Related KEYWORDS: **#%, #$**

BREDEN'S BASIC

TRAP

Type:	Error trapping command
Format:	TRAP <line number>
Example:	TRAP 1000
Action:	The computer will go to line 1000 when an error occurs.
Parameters:	<line number> AEXP : 0 to 63999
Explanation:	The TRAP command is used to make the computer go to a specified line when an error occurs. Then, if need be, the type of error and line where it occurred can be requested. It makes sense to put TRAP commands into your program BEFORE any parts which are likely to cause errors.

Related KEYWORDS: **ERRL, ERRN, HELP**

ERRN

Type: Error trapping command

Format: ERRN

Example: PRINT ERRN

Action: The error number where the error occurred is displayed.

Parameters: None.

Explanation: The ERRN function is used to obtain a value representing the type of error that occurred. Please note that it is possible to trap errors relating to Commodore BASIC as well as errors relating to BREDEN's BASIC. All of Commodore's errors have values LESS than 128, whereas all of BREDEN's errors have values GREATER than 128. Consult the following table for the error relating to the error number. The ERRN function can be extremely useful when trapping unexpected errors during program execution. Also see the ERRL and TRAP commands which complement ERRN.

Error number	Type of error		
1	Too many files	20	Division by zero
2	File open	21	Illegal direct
3	File not open	22	Type mismatch
4	File not found	23	String too long
5	Device not present	24	File data
6	Not input file	25	Formula too complex
7	Not output file	26	Can't continue
8	Missing file name	27	Undef'd function
9	Illegal device number	28	Verify
10	Next without For	29	Load
11	Syntax	30	Break
12	Return without gosub		
13	Out of data	129	Until
14	Illegal quantity	130	>31 repeats/loops
15	Overflow	131	>63 defined
16	Out of memory	132	>31 perfs
17	Undef'd statement	133	Perf not found
18	Bad subscript	134	Finish
19	Redim'd array		

Related KEYWORDS: **ERRL, TRAP, HELP**

ERRL

Type:	Error trapping function
Format:	ERRL
Example:	PRINT ERRL
Action:	The line number where the error occurred is displayed.
Parameters:	None.
Explanation:	The ERRL function is used to obtain a value representing the program line number where an error occurred. This can be extremely useful when trapping unexpected errors during program execution. Also see the ERRN and TRAP commands which complement ERRL.

Related KEYWORDS: **ERRN, TRAP, HELP**

HELP

Type:	Error trapping command
Format:	HELP
Example:	HELP
Action:	The computer will display the line where the error occurred and will place a solid square where the error is.
Parameters:	None.
Explanation:	The HELP command is used when you are literally shouting for help! See the example above for an explanation of what HELP will do for you.

Related KEYWORDS: **ERRL, ERRN, TRAP**

KEY

Type:	Programming aid command
Format:	KEY <key number>,[group of characters]
Example:	KEY 1,"list←"
Action:	The word 'list' and a carriage return are assigned to function key 1.
Parameters:	<key number> AEXP : 1 to 16 [group of characters] SEXP : up to 63 characters

Explanation:

The KEY command is used to assign a group of characters to a function key. This is done so that by merely pressing one key a whole expression may be printed on the screen and/or executed. This has the advantage that once a particular function key has been defined, just by pressing one key, a particular expression ("list" for example) may be printed. This can save a lot of time when using a keyword (LIST or REPEAT for example) often during a programming session. A special feature of the KEY command is the use of the back-arrow symbol ("←"). When this symbol is placed at the end of the group of characters, pressing the function key in question will actually execute the group of characters as well as printing them on the screen. This makes it unnecessary to press the RETURN key. Pressing the function keys with or without the SHIFT key allows you to reach function keys 1 to 8. If you also press the CTRL key, this will have the effect of adding 8 to the function key number. For example, CTRL F1 gives you function key 9 (8 plus 1), CTRL SHIFT F2 gives you function key 10 (8 plus 2). In this way it is possible to reach all 16 function keys.

Related KEYWORDS: **SKEYS, MEMLOAD, DISPKEYS**

DISPKEYS

Type:	Programming aid command
Format:	DISPKEYS
Example:	DISPKEYS
Action:	A list of the words assigned to the function keys is displayed.
Parameters:	None.
Explanation:	The DISPKEYS command is used to display a list of the words assigned to the function keys. This is useful if you have forgotten what is on a particular function key. Another useful feature of this command is that once you have the list of function keys on the screen, it is very easy to move the cursor up to a particular line to define or redefine a key, similar to the way in which a BASIC program line is edited.

Related KEYWORDS: **KEY, SKEYS**

SKEYS

Type:	Programming aid command
Format:	SKEYS [File name],<device number>
Example:	SKEYS "keys.sound",1
Action:	The words on the function keys are saved onto tape with a name of 'keys.sound'
Parameters:	[File name] SEXP <device number> AEXP
Explanation:	The SKEYS command is used to store the words you have programmed onto the function keys. This is a good idea if you use a particular set of keywords regularly (LIST, RUN, DISPKEYS for example). The next time you wish to use the same set of keywords on the function keys, all you have to do is use the MEMLOAD command to bring back the stored words.

Related KEYWORDS: **MEMLOAD, KEY, DISPKEYS**

MEMLOAD

Type:	Programming aid command
Format:	MEMLOAD [File name],<device number>
Example:	MEMLOAD "keys.graphics",8
Action:	A program or bytes called 'keys.graphics' is LOADed.
Parameters:	[File name] SEXP <device number> AEXP

Explanation: The MEMLOAD command is used to LOAD machine code or any other bytes (function key data, sprite data or udg character sets for example). The advantage MEMLOAD has over the conventional LOAD command is that it restores the BASIC memory pointers to their original state. If bytes are to be LOADed into the area of memory from $0C00 to $8000 (3072 to 32768 decimal) it is a good idea to set HIMEM to a suitable limit to prevent variables used within BASIC programs overwriting the bytes LOADed in.

Related KEYWORDS: **MEMSAVE, LOAD, SAVE**

MEMSAVE

Type: Programming aid command

Format: MEMSAVE [File name],<device number>,<start address>,<end address>

Example: MEMSAVE "sort code $7a00",8,# $7a00,# $7cd0

Action: Memory from location $7a00 to $7cd0 is SAVEd onto disk with a name of 'sort code $7a00'

Parameters:

[File name]	SEXP
<device number>	AEXP
<start address>	AEXP
<end address>	AEXP

Explanation: The MEMSAVE command is used to SAVE machine code or any other bytes (sprite data for example). This command is of particular use to the experienced BASIC and/or machine language programmer. Please note the use of hexadecimal notation when giving the start and end addresses in the example given above. This, of course, is optional.

Related KEYWORDS: **MEMLOAD, LOAD, SAVE**

FIND

Type:	Programming aid command
Format:	FIND ←[search string]←
Example:	FIND ←POKE←
Action:	All the line numbers which contain the POKE command will be displayed on the screen.
Parameters:	[search string] SEXP
Explanation:	The FIND command is used to make the computer search for a particular phrase used within your program (PRINT for example). When the computer finds any occurrences of the phrase, it will display the line number(s). Please note that if something within speech marks is to be searched for, replace the '←' symbol at either end of the search string with a speech mark. This is for technical reasons known as 'tokenisation'.

Related KEYWORDS: **None.**

ERASE

Type:	Programming aid command
Format:	ERASE <first line number>,<last line number>
Example:	ERASE 1000,1040
Action:	All lines are removed from 1000 to 1040 INCLUSIVE.
Parameters:	<first line number> AEXP : 0 to 63999 <last line number> AEXP : 0 to 63999
Explanation:	The ERASE command is used to remove unwanted lines in a BASIC program. This command will remove all lines inclusive of the first and last line specified, so use this with caution. As a protective measure, this command will abort with an error message if either the first and/or last line is not present, without removing any lines.

Related KEYWORDS: **None.**

OLD

Type:	Programming aid command
Format:	OLD
Example:	OLD
Action:	A NEWed program is now recovered.
Parameters:	None
Explanation:	The OLD command is used to recover a BASIC program that has accidentally been NEWed. This is useful if the user has inadvertently typed 'NEW'. Please note that usually it will be impossible to recover a program if an error message has been generated (?SYNTAX ERROR for example) after NEWing a program.

Related KEYWORDS: **NEW**

RKEY

Type:	Programming aid command
Format:	RKEY <repeat type>
Example:	RKEY 3
Action:	All keys now repeat when held down.
Parameters:	<repeat type> AEXP : 1 to 3 1 : only cursor keys repeat 2 : no keys repeat 3 : all keys repeat
Explanation:	The RKEY command is used to make some, all, or none of the keys on the computer's keyboard repeat when held down.

Related KEYWORDS: **None.**

HLIGHT

Type:	Programming aid command
Format:	HLIGHT <switch : on/off>
Example:	HLIGHT 1
Action:	Highlighting facility is turned on.
Parameters:	<switch : on/off> AEXP : 1 or 0

Explanation: The HLIGHT command is used to turn on/off the facility to highlight commands within a BASIC program that are specific to BREDEN's BASIC. After 'HLIGHT 1' has been typed and the program is listed, all BREDEN's BASIC commands are highlighted. This can improve the appearance of a program listing, especially on a printer (although see the note in your printer manual on prolonged printing of REVERSE characters). If you intend your BASIC program to be sold commercially, use of the HLIGHT command will show you which commands you will have to convert into standard Commodore BASIC before selling.

Related KEYWORDS: **LIST**

ALPHA

Type:	Programming aid command
Format:	ALPHA <switch: on/off>
Example:	ALPHA 1
Action:	'quick keyword entry' mode is turned on.
Parameters:	<switch: on/off> AEXP : 1 or 0
Explanation:	The ALPHA command is used to turn on or off the 'quick keyword entry' mode. When this mode is on, pressing alphabetic keys (a-z) in conjunction with either the SHIFT or COMMODORE logo key, will cause the computer to print out the relevant keyword assigned to the key pressed. This can be a useful time saver once you have learned which keys the keywords you use most often are on. When this mode is off, the usual graphic symbol will be displayed. For a complete list of the predefined keywords, please refer to the appendices.

Related KEYWORDS: **None.**

WHILE...DO...ELSE...

Type:	Structured programming
Format:	WHILE <condition> DO <if true>:ELSE <if false>
Example:	10 WHILE FKEY<>1 DO GOTO 10:ELSE PRINT"pow !"
Action:	This loops until function key 1 is pressed, when the message 'pow !' will be displayed.
Parameters:	None.
Explanation:	The structure WHILE...DO...ELSE... is one of the most powerful techniques in assessing the truth or falsity of a given condition. In the case above, for example, the condition is whether the value of the function key pressed is unequal to 1 (FKEY<>1). After the WHILE command, the condition to be checked for must be stated (FKEY<>1). After the condition, there must be a DO command. If the result of the condition is true, the code after the DO command is executed, otherwise, the code after the ELSE command (if there is one) is executed. Please note that if you are already acquainted with the IF...THEN... structure you will find usage of this new structure fairly straightforward, although particularly useful.

Related KEYWORDS: **IF, THEN**

REPEAT...UNTIL...

Type:	Structured programming
Format:	REPEAT...UNTIL <condition>
Example:	REPEAT x=x+1:PRINT x;:UNTIL x=20
Action:	This prints numbers consecutively from 1 to 20.
Example:	REPEAT UNTIL FALSE
Action:	This loops indefinitely.
Parameters:	None.
Explanation:	REPEAT...UNTIL is a powerful LOOP structure. It is functionally similar to the FOR...NEXT loop structure with the exception that instead of specifying the number of loop iterations at the START of the loop, the number of loop iterations is determined by the condition test at the END of the loop (after the UNTIL command). When the condition after the UNTIL command (x=20, for example) is false (i.e. x is not 20) the loop is executed again. Only when the condition is true (i.e. x=20) does the program leave the REPEAT...UNTIL loop. Please note that never ending loops are possible through use of the FALSE command in conjunction with REPEAT...UNTIL structures. There is an example of this in the second example above. A maximum of 31 REPEAT...UNTIL loops may be nested.

Related KEYWORDS: **LOOP**

LOOP...UNTIL...

Type:	Structured programming
Format:	LOOP...UNTIL <condition>
Example:	LOOP x=x+1:PRINT x;:UNTIL x=20
Action:	This prints numbers consecutively from 1 to 20.
Example:	LOOP UNTIL FALSE
Action:	This loops indefinitely.
Parameters:	None.
Explanation:	LOOP...UNTIL is a powerful LOOP structure. It is functionally identical to the REPEAT...UNTIL structure. However, in certain cases, it may be more meaningful to actually use the word 'LOOP' than the word 'REPEAT'. It is functionally similar to the FOR...NEXT loop structure with the exception that instead of specifying the number of loop iterations at the START of the loop, the number of loop iterations is determined by the condition test at the END of the loop (after the UNTIL command). When the condition after the UNTIL command (x=20, for example) is false (i.e. x is not 20) the loop is executed again. Only when the condition is true (i.e. x=20) does the program leave the LOOP...UNTIL loop. Please note that never ending loops are possible through use of the FALSE command in conjunction with LOOP...UNTIL structures, as can be seen in the second example given above. A maximum of 31 LOOP...UNTIL loops may be nested.

Related KEYWORDS: **REPEAT**

SEARCH

Type:	Structured programming
Format:	SEARCH
Example:	SEARCH
Action:	This searches for and stores all Performance definitions.
Parameters:	None.
Explanation:	The SEARCH command is used to make the computer search for and 'remember' all PERF...FINISH structures (Performances). It is good programming practice to place this command right at the beginning of a program which uses Performances. Note that if too many (more than 63) Performances are defined, the program will abort with the message '>63 defined'.

Related KEYWORDS: **PERF, FINISH**

PERF

Type:	Structured programming
Format:	PERF [name of Performance]
Example:	PERF "Read joystick 1"
Action:	This searches for a Performance called "Read joystick 1" and, if found, executes the code found there.
Parameters:	[name of Performance] SEXP : up to 28 characters

Explanation:

The PERF command is used to perform sections of code which may be called by name. This means that you may split your program into sections which perform a specific task (called Performances) such as reading a joystick (as in the example above) or sorting a list of names etc. This will improve the readability of your programs a great deal, and also by simply looking at the Performance name it should now be possible to determine what the Performance does. The trouble with the GOSUB command is that simply by looking at 'GOSUB 1000' within a program, it is not possible to tell what that subroutine does, and when you come to modify the program (say a month later), it can be a nightmare trying to remember what the subroutines actually do! Because of the nature of this programming structure, it is faster than normal GOSUB...RETURN structures. With an 8K program, a PERF...FINISH structure was 7.19 times FASTER than a GOSUB...RETURN structure performing exactly the same task. That's FAST! Note that as string expressions may be used to represent the name of a Performance, powerful 'PERFa$(x)' structures are possible, allowing for extremly versatile branching. For example, the variable 'x' may contain the option number chosen by the user from a menu. The array 'a$( )' may contain a whole list of Performance names. Thus, 'x' may select the subscript of the array which in turn determines which Performance name is used. See example below:

```
10   SEARCH
20   LOOP C=LSB(C+1)
30   PERF "COLOUR"
40   UNTIL FALSE
50 :
1000 ["COLOUR"]
1010 BCOL C
1020 FINISH
```

Related KEYWORDS: **SEARCH, FINISH**

FINISH

Type:	Structured programming
Format:	FINISH
Example:	FINISH
Action:	This causes the program to return from a Performance.
Parameters:	None.
Explanation:	The FINISH command is used to terminate execution of a Performance. It is placed at the END of a Performance (section of code) and instructs the computer to leave that part of BASIC code and to return to the next instruction after the PERF command, rather like the RETURN command does after a GOSUB.

Related KEYWORDS: **SEARCH, PERF**

	POP
Type:	Enhanced programming command
Format:	POP
Example:	POP
Action:	The 'RETURN address' is removed off the STACK.
Parameters:	None.
Explanation:	The POP command is used to exit subroutines. When a GOSUB command has been issued and, for one reason or another, you do not want to RETURN to the main part of the program, a POP command MUST be used in order to tell the computer what you are doing. If POP is not used in circumstances such as the one mentioned, unexpected '?OUT OF MEMORY' errors may occur.

Related KEYWORDS: **GOSUB, RETURN**

HIMEM

Type:	Reserved system variable
Format:	Use exactly as you do with a normal variable
Example:	PRINT HIMEM
Action:	The first address ABOVE BASIC memory is printed.
Example:	HIMEM=28672
Action:	This sets the LAST address used by BASIC to 28671.
Parameters:	None.
Explanation:	HIMEM is used exactly like other variables. The examples above show how it can be used. Please note that you should use HIMEM to set the last address used by BASIC before you LOAD machine code or bytes into the area normally used by BASIC (memory from $0c00 to $8000). Then you should use the MEMLOAD command. If this procedure is not adhered to, you may be confronted with unexpected '?OUT OF MEMORY ERROR' messages. The above procedure also prevents BASIC from corrupting the machine language program loaded with its variables.

Related KEYWORDS: **CALL, MEMLOAD, MEMSAVE**

CALL

Type: Enhanced programming command

Format: CALL <address of machine language>,<value>,<value>,...

Example: CALL ♯ $7000,X,100+SIN(X/20)★50

Action: Control is passed to a machine language program at address 28672 and the parameters X and 100+SIN(X/20)★50 are stored as 2 byte integers sequentially in memory from address $033C onwards. (Don't try it now, as there is no machine language there at the moment!)

Parameters:

<address of machine language>	AEXP : 0 to 65535
<value>	AEXP : 0 to 65535

Explanation: CALL is used to transfer control to a machine language program you may have written. Any numeric expressions may be passed to your machine language as long as the result is positive and less than 65536. The parameters you send to a machine language program will be stored in normal low byte, high byte order in memory from location $033C upwards. If you need to, you can even ask the computer how many parameters were passed to it. This is done by doing the equivalent of PEEK(2) in machine code (LDA $02 for example). Please note that parameter passing to your machine code is entirely optional. Finally, at the end of your machine code, you MUST use an 'RTS' instruction.

Related KEYWORDS: **HIMEM, MEMLOAD, MEMSAVE**

PAUSE

Type:	Enhanced programming command
Format:	PAUSE <time delay>
Example:	PAUSE 5
Action:	The computer pauses for half a second.
Parameters:	<time delay> AEXP : 1 to 255
Explanation:	The PAUSE command is used to make the computer pause for a given time. The time delay specified is in tenths of a second, thus quite short and accurate delays are possible. The delay given in the example would make the computer pause for half a second (5/10=1/2). This command is especially useful when you need to slow the computer down in order to see what it is doing.

Related KEYWORDS: **None.**

RLINE

Type:	Enhanced programming command
Format:	RLINE <line number>
Example:	RLINE 1000
Action:	The next item of DATA will be taken from line 1000.
Parameters:	<line number> AEXP : 0 to 63999
Explanation:	The RLINE command is used to make the computer go to the line number specified for its DATA when the next READ command is issued. You could have question and answer DATA related to different topics at line 1000 onwards, the line number of each different DATA block being greater than the last by a certain amount (say 100). Then you could READ any particular DATA block by using a formula. For example, 1000 + (data block number ★ 100) would calculate the right line number for any block if the above method is adopted. Having calculated the correct line number for a particular DATA block, use RLINE, e.g. RLINE <line number>

Related KEYWORDS: **DATA, READ, RESTORE**

PREDEFINED KEYWORDS ON ALPHABETIC KEYS

There are 52 predefined keywords assigned to the alphabetic keys (a-z). After entering 'quick keywords entry' mode (see ALPHA), simply by holding down the SHIFT or Commodore key in conjunction with an alphabetic key, it is possible to make a whole keyword be displayed instantly, thus saving typing. This is only useful once you have learned which keywords are assigned to which keys. To help you with this, the following table is provided.

Key	Commodore key	SHIFT key
a	ABS	ASC
b	AND	RND
c	CHR$	COS
d	DEF	DEF
e	END	EXP
f	FN	FOR
g	GOSUB	GOTO
h	GET	POKE
i	INPUT	IF
j	CLOSE	OPEN
k	PRINT	INPUT
l	LEFT$	LEN
m	MID$	MID$
n	NEW	NEXT
o	OR	ON
p	PEEK	PRINT
q	TAB(	INT
r	RIGHT$	RETURN
s	STR$	SIN
t	TAN	THEN
u	USR	USR
v	VAL	VAL
w	LOG	ATN
x	SGN	SQR
y	REM	POS
z	SYS	CLR

DISK COMMANDS

There are many commands which may be given to the disk drive via the DOS command. They range from formatting a disk to copying files. If you are in any doubt whatsoever about the effect of any of the commands shown below, please consult your disk drive owner's guide for further details.

BACKUP
DOS "D1=0"
This makes an identical copy of a disk in drive 0 onto a disk in drive 1 (dual drive disk units only).

COLLECT (VALIDATE)
DOS "V0"
This re-organises a disk that has been in use for some time so that it yields the maximum amount of work space. Also, this command will collect all blocks taken up by files which have never been CLOSEd properly. Never use this command on a disk containing random files.

CONCAT
DOS "CO:result=0:file1,0:file2"
This creates a file called 'result' which contains the contents of 'file1' and 'file2' (the two files joined together, or CONCATenated).

COPY
DOS "C1=0"
This copies all files from drive 0 onto the end of the list of files on a disk in drive 1 (dual drive disk units only).

DOS "CO:char2=char1"
This makes a duplicate copy of a file called 'char1' on drive 0, but with a name of 'char2'.

DOS "C1:picture=0:picture"
This copies a file called 'picture' from drive 0 to drive 1 (dual drive disk units only).

HEADER (FORMAT)
DOS "No:picture disk,pd"
This formats a disk in drive 0 with a name of 'picture disk' and an identifier or 'pd'. Use with EXTREME CARE!!!

INITIALISE
DOS "I0"
This initializes a disk in drive 0.

RENAME
DOS "R0":chemistry=chem"
This renames a file on drive 0 from 'chem' to 'chemistry'.

SCRATCH (DELETE)
DOS "SO:spihcorcim"
This SCRATCHes or deletes a file on drive 0 called 'spihcorcim'. Use with care!

STORING AND RECALLING GRAPHICS SCREENS

This appendix shows you how to store a picture on the graphics screen to disk or tape, and then how to recall that picture for viewing at a later date. The examples shown are for disk, but in order for them to work with cassette, simply replace the '8' with a '1'.

STORING A GRAPHIC SCREEN

Example:
SHIRES "sine curve",8
This will store 2 files, which are called:
'sine curve 1', and
'sine curve 2'.
Note that the file name of any picture stored will always be 16 characters long.

RECALLING A GRAPHIC SCREEN

Example:
MEMLOAD "sine curve 2",8
COPY
MEMLOAD "sine curve 1",8"
The above sequence of commands will recall a graphic screen from disk and will store it in memory, ready for your use.
Now, to display the graphic screen recalled, simply type the following:

GRAPHICS: PAUSE 10

The pause may be for longer if you wish. Please note that although the above sequence for recalling a graphic screen was executed in direct mode, it could have been executed from within a program.

STORING AND RECALLING UDG CHARACTER SETS

This appendix shows how you store a UDG character set to disk or tape, and then how to recall that character set for use at a later date. The examples shown are for disk, but in order for them to work with cassette, simply replace the '8' with a '1'.

STORINGA UDG CHARACTER SET

Example:
SUDG "compufont",8
This will store a file called 'compufont' onto disk.

RECALLING A UDG CHARACTER SET

Example:
MEMLOAD "compufont",8
UDG
COPY
The above sequence of commands will recall a UDG character set from disk and will allow immediate use of them.
Please note that although the above sequence for recalling a UDG character set was executed in direct mode, it could have been executed from within a program.

CREATING A USER DEFINED GRAPHIC CHARACTER (UDG)

This appendix shows you how to create UDG characters of your own. If you want to store them onto tape or disk, see appendix d. By the end of this appendix, you may appreciate that designing your own UDGs is a lengthy process for more than just a few characters. If you would like a tool for creating UDGs quickly and effortlessly, then watch out for the 'UDG EDITOR' program from VISIONS.

MARK OUT THE SHAPE

Example:

bit value 1 2631 84268421	pattern value
!★ ★....★ ★!	195 (128+64+2+1)
!★ ★....★ ★!	195 (128+64+2+1)
!.....!	0
!..★ ★ ★ ★..!	60 (32+16+8+4)
!..★ ★ ★ ★..!	60 (32+16+8+4)
!.....!	0
!★ ★....★ ★!	195 (128+64+2+1)
!★ ★....★ ★!	195(128+64+2+1)

STORE THE PATTERN VALUES

Once the computer is in UDG mode (see UDG), this little program below will store a UDG for you, given the pattern values.

```
10 FOR X=1 TO 8
20 READ A
30 A$=A$+CHR$(A)
40 NEXT x
50 SETUDG 0,A$
60 DATA 195,195,0,60,60,0,195,195
```

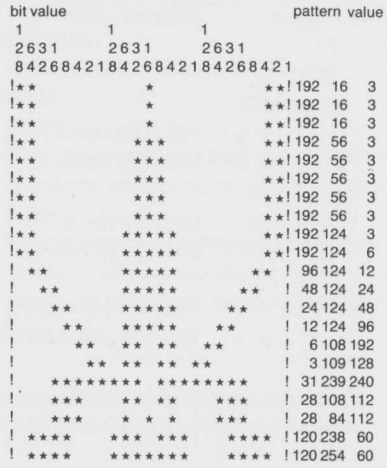
The '0' after SETUDG in line 50 of the above program can be replaced with the poke code (o to 255) of any character. Good luck with designing your own UDGs!

CREATING A SPRITE (MOVEABLE OBJECT BLOCK, OR MOB)

This appendix shows you how to create your own sprites. If you want to store them onto tape or disk, see appendix g. By the end of this appendix, you may appreciate that designing your own sprites is a lengthy process. If you would like to be able to create sprites quickly and effectively, then look out for the 'SPRITE EDITOR' program from VISIONS.

MARK OUT THE SHAPE

Example:



STORE THE PATTERN VALUES

The short program below will store the shape for a sprite, given the pattern values.

```
10 FOR X=1 TO 63
20 READ A
30 A$=A$+CHR$(A)
40 NEXT X
50 MOBSLOT 1,A$
60 SETMOB 1,1,0,1,1,0,5,5,5,1
70 DATA (place pattern values here)
```

If you have followed the above procedure correctly, you will now be able to move the sprite around the screen. Type:

The sprite should appear at a position 200 across, 100 down. Good luck with your own sprites, it's good fun!

STORING AND RECALLING SPRITES

This appendix shows you how to store the sprites stored in memory onto tape or disk, and then how to recall sprites stored on tape or disk. The examples shown are for disk, but in order for them to work with cassette, simply replace the '8' with a '1'.

STORING SPRITES

Sprites are stored in two separate blocks in memory. The first block, from 1024 to 2047 ($0400 — $07FF), is used for displaying sprites on the text screen. The second block, from 50176 to 51199 ($C400 — $C7FF), is used for displaying sprites on the graphics screen. Thus, if you store block 1 only, you will only be able to display sprites on the text screen at a later date. Similarly, if you store block 2 only, you will only be able to display sprites on the graphics screen at a later date. Therefore, if you want to be able to display sprites on either the text screen or the graphics screen, you MUST store BOTH sprite blocks.

STORING BLOCK 1

Example:
MEMSAVE "sprites 1",8, ♯ $0400, ♯ $0800

STORING BLOCK 2

Example:
MEMSAVE "sprites 2",8, ♯ $C400, ♯ $C800

RECALLING SPRITES

As mentioned above, there are 2 sprite blocks. Depending on which block was stored (unless both were), you will only be able to display sprites on one screen, either the text screen or the graphics screen. Regardless of which sprite block was stored, the method of recalling the sprites is the same.

Example:
MEMLOAD "sprites",8

MULTICOLOUR CHARACTER MODE

This appendix shows you how to use multicolour character mode. This mode is selected by typing 'GMODE 1'. This should be typed when in text or UDG mode. However, it is most useful in UDG mode, as standard Commodore characters are not designed for use in multicolour mode. When designing UDGs for use in multicolour mode, note that 2 bits are used to represent 1 VISIBLE pixel. Thus, this means the horizontal resolution of UDGs is halved from 8 pixels to 4 pixels. The advantage of using multicolour character mode is that 4 colours may be used for each UDG. As you may or may not know, 2 bits allow 4 possible bit combinations. They are:

bit pair	where colour comes from
00	screen colour (see SCOL)
01	1st colour of MCOLS (see MCOLS)
10	2nd colour of MCOLS (see MCOLS)
11	colour memory (see DCOL or CBM manual)

Example multicolour UDG:

bit value	pattern value
1	
2 6 3 1	
8 4 2 6 8 4 2 1	
! ★ ★ ★ ★!	85 (bit pair '01')
! ★ ★ ★ ★!	85
! ★ ★ ★ ★!	85
!★ ★ ★ ★ !	170 (bit pair '10')
!★ ★ ★ ★ !	170
!★ ★ ★ ★ !	170
!★★★★★★★★!	255 (bit pair '11')
!★★★★★★★★!	255

The following example program will illustrate this:

```
1 UDG
2 A$=CHR$(85):B$=CHR$(170):C$=CHR$(255)
3 SETUDG 0,A$+A$+A$+B$+B$+B$+C$+C$
10 GMODE 1
20 CLS
50 X=0:LOOP X=X+1
60 PRINT"@ @ @ @ @ @ @ @ @ @"
65 PRINT
70 UNTIL X=10
100 REPEAT
110 MCOLS IRAN(1,16),IRAN(1,16)
120 DCOL IRAN(1,8)+8
130 PAUSE 5
140 UNTIL FALSE
```

EXTENDED BACKGROUND COLOUR MODE

This appendix shows you how to use extended background colour mode. This mode is selected by typing 'GMODE 2'. This should be typed when in text or UDG mode. As the name of this mode (extended background colour) suggests, this mode controls the background colour of characters on the screen. Colour memory (see DCOL) controls the foreground colour of characters on the screen. Note that, because of the nature of this mode, only character definitions with values (poke codes) of between 0 and 63 can be displayed. The value of the character displayed controls where the background colour comes from. See the table below.

Character value	Where colour comes from
0-63	screen colour (see SCOL)
64-127	1st colour of ECOLS (see ECOLS)
128-191	2nd colour of ECOLS (see ECOLS)
192-255	3rd colour of ECOLS (see ECOLS)

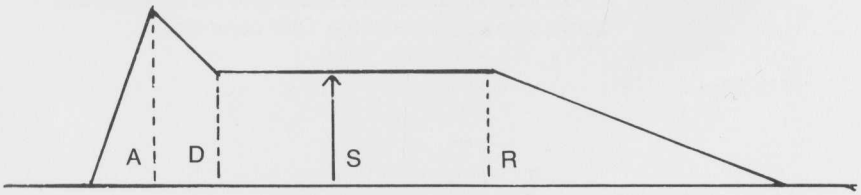
The program below illustrates this:

```
1 CLS
2 FLICKER 0
5 DEF FNC(X)=IRAN(1,16)
10 GMODE 2
15 ECOLS 2,6,1
20 FOR R=0 TO 24
30 FOR C=0 TO 39
40 PUT ♯ $0800+R★40+C, MOD(R,4)★64+1
60 NEXT C,R
70 :
80 LOOP
82 SCOL FNC(X)
84 DCOL FNC(X)
86 ECOLS FNC(X), FNC(X), FNC(X)
88 PAUSE 5
90 UNTIL FALSE
```

ERROR MESSAGES	
?? UNTIL ERROR	This error is displayed when an 'UNTIL' command has been used without a 'REPEAT' or 'LOOP' command.
??>31 REPEATS/ LOOPS ERROR	This error is displayed when more than 31 'Repeat...Until' or 'Loop...Until' structures have been nested. As this is unusual, the most likely cause is that an 'UNTIL' command has been repeatedly omitted.
??>63 DEFINED ERROR	This error is displayed when more than 63 Performances have been defined.
??>31 PERFS ERROR	This error is displayed when more than 31 'PERF "NAME"' structures have been nested. As this is unusual, the most likely cause is that the 'FINISH' command has been omitted at the end of a Performance.
??PERF NOT FOUND ERROR	This error is displayed when a Performance name has been specified with the 'PERF' command and it cannot be found in the BASIC program.
??FINISH ERROR	This error is displayed when a 'FINISH' command is executed without previous usage of the 'PERF' command. This is similar to Commodore's own '? RETURN WITHOUT GOSUB ERROR'.

SOUND ENVELOPE CRITERIA

To allow you control over how each note, or frequency, is played, you can specify different vlues for the attack, decay, sustain and release.



A=attack (time)
D=decay (time)
S=sustain (volume)
R=release (time)

ATTACK Attack determines the rate at which a note or frequency reaches .naximum volume from zero volume.

DECAY Decay determines the rate at which a note or frequency falls (or decays) from maximum volume to the sustain volume.

SUSTAIN Sustain is the volume at which a note or frequency is held.

RELEASE Release determines the rate at which a note or frequency falls (or decays) from the sustain volume to zero volume.

SPECIAL FEATURES OF BREDEN'S BASIC

HALTING EXECUTION

Execution of any program or action currently being carried out by the computer can be paused, immediately by holding down both the '←' and '1' keys together, '1' first.

HALTING 'LIST' or 'DIR'

During a program LIST to the screen, pressing of either the SHIFT or SHIFT LOCK keys will temporarily halt the LISTing to the screen. Releasing the SHIFT or SHIFT LOCK key will resume the LISTing to the screen. The above also applies with the 'DIR' command.

Visions

BREDEN'S BASIC

BREDEN'S **BASIC** is *the* extended **BASIC** language for the Commodore 64 computer. Simple to use, yet awesome in its power, BREDEN'S **BASIC** sports over one hundred and thirty extra programming commands covering many previously unsupported areas.

These areas include:
COLOUR HIGH RESOLUTION GRAPHICS
MULTICOLOUR SPRITE GRAPHICS
MUSIC AND SOUND SYNTHESIS
STRUCTURED PROGRAMMING
USER DEFINED GRAPHICS (UDGs)
TOOLKIT FACILITIES
PLUS MANY OTHERS

BREDEN'S **BASIC** can be used by anyone, preferably with a grounding of standard Commodore **BASIC.**
For the more experienced programmer, writing in machine language, BREDEN'S **BASIC** offers much in the way of interfacing machine language and **BASIC** to form a powerful combination.
For those who occasionally have reason to cry out **"HELP!"** when a program fails to work in the correct or intended way, it is now possible to type **HELP** into the computer. The computer will then show where the problem lies!

£

DESIGNED AND PRODUCED BY DEREK ASLETT ASSOCIATES